INF-253 Lenguajes de Programación

Profesores: José Luis Martí Lara, Wladimir Ormazabal Orellana
Ayudante coordinador: Bastián Ríos Lastarria
Ayudantes: Fernanda Araya, Diego Duarte, Andrés Jablonca, Cristian Tapia

25 de mayo de 2025

1. Introducción

¡Bienvenido/a emprendedor/a!
Tras tu meteórico paso por la torre competitiva Javaling, has decidido invertir tus recién ganados 100 000 000 CLP en un restaurante robotizado de alta gama llamado **'Lambda Lab'**, utilizando tus conocimientos adquiridos como Administrador de contraseñas y trabajador en McSansano.
En este laboratorio gastronómico, brazos robóticos preparan y emplatan combos de degustación millimétricos, mientras tú supervisas la operación mediante un sistema escrito en Scheme y bajo un estricto estilo puramente funcional.
Para la gran inauguración en la Feria de Emprendimiento Sansano, la franquicia exige que automatices cuatro procesos críticos y demuestres tu dominio de las funciones lambda y la recursión.

2. Funciones a implementar

P1. Conicide?

Un **brazo robótico** novato realiza un escaneo inicial de cada bandeja de ingredientes y anota su conteo estimado. Sin embargo, sus sensores a veces fallan. Necesitas una función que valide rápidamente si el número registrado coincide con la cantidad real de elementos.

- **Sinopsis:** (coincide? n ls)
- **Característica Funcional:** Listas simples y recursión simple.
- **Descripción:** Se recibe un número entero no negativo n y una lista ls. La función debe devolver #**t** si n coincide con la cantidad real de elementos de ls y #**f** en caso contrario.
    - No se permite usar ***length, list-ref, member, for-each*** ni ninguna construcción imperativa.
    - Debe implementarse mediante recursión simple.
    - La función debe funcionar para todo n ≥ 0 y listas vacías.

`*en caso de dar problemas el nombre 'coincide?' se permite cambiarlo a solo 'coincide'`

Ejemplo:

```
>(coincide? 3 '(pan ketchup lechuga))
#t
>(coincide? 0 '())
#t
>(coincide? 2 '(hamburguesa queso tomate))
#f
```

P2. Mezclador Gourmet

Un **robot–mezclador** se encarga de combinar, paso a paso, los ingredientes de cada pedido. Para modelar este proceso programarás dos versiones de un *plegado* (*fold*) sobre listas: una clásica (*simple*) y otra optimizada en cola.

- **Sinopsis:**
  `(mezclador-gourmet-simple f inicio ls)`
  `(mezclador-gourmet-cola  f inicio ls)`
- **Característica Funcional:** Listas simples, recursión simple y recursión de cola.
- **Descripción:**

  1. `f` : función de 2 argumentos (`(lambda (acum elem) ...)`.
  2. `inicio` : valor inicial del "plato" en construcción.
  3. `ls` : lista de ingredientes o valores a procesar.

  Cada versión recorre `ls` aplicando `f` sucesivamente, de modo que el resultado final equivale a

  $$f(x_n,\ f(x_{n-1},\ \ldots f(x_1,\ \mathit{inicio})\ldots)).$$

  - `mezclador-gourmet-simple` debe implementarse con recursión *simple*.
  - `mezclador-gourmet-cola` debe ser con **recursión de cola**: la llamada recursiva debe quedar como *última* operación.
  - Está prohibido el uso de las siguientes funciones (`foldl`, `foldr`, `for-each`, `length`, etc.). (ver **sobre la entrega** para más detalles)

Ejemplos:

```
>(mezclador-gourmet-simple + 0 '(8000 12000 15000))
R:35000
>(mezclador-gourmet-cola  + 0 '(8000 12000 15000))
R:35000
>(mezclador-gourmet-simple * 1 '(2 3 4))
R:24
>(mezclador-gourmet-cola  * 1 '(2 3 4))
R:24
```

Un ejemplo más complejo para que se entienda correctamente el funcionamiento: Para los siguientes casos, considere 'f' como la funcion lambda, inicio como '() y 'ls' seria '(pan lechuga tomate queso) para ambos casos.

```
;; torre de sandwich version simple y de cola
>(mezclador-gourmet-simple
  (lambda (ing pila) (cons ing pila))
  '()
  '(pan lechuga tomate queso))
R:'(pan lechuga tomate queso)
;; Pila de capas invertida
>(mezclador-gourmet-cola
  (lambda (pila ing) (cons ing pila))
  '()
  '(pan lechuga tomate queso))
R:(queso tomate lechuga pan)
```

Ten en cuenta que en los primeros ejemplos la suma y la multiplicación no cambian el orden porque son operaciones conmutativas, pero al simular la torre de sándwich con la lista (pan lechuga tomate queso), se puede apreciar la diferencia entre ambas versiones: la versión simple se evalúa "de adentro hacia afuera" —procesa primero el final de la lista— y al subir de la recursión antepone cada ingrediente, manteniendo el orden original, mientras que la versión de cola recorre la lista de izquierda a derecha y, al anteponer cada elemento al acumulador en cada paso, produce la secuencia invertida.

P3. Secuencia-Rotacional

Un **circuito rotacional** de preparación recibe una lista de "estaciones" (funciones unarias) y un flujo de insumos. Donde primero, se rota la lista de funciones un paso a la izquierda, luego se aplica la primera función al acumulador y al siguiente insumo, se guarda el nuevo acumulador en la lista de resultados y luego se repite recursivamente hasta procesar todos los insumos.

- **Sinopsis:**`(secuencia-rotacional fs inicio xs)`
- **Característica Funcional:** Listas simples, expresiones lambda, recursión simple.
- **Descripción:**

  1. `fs` : lista de funciones unarias
     (cada estación aplica un paso de la receta).
  2. `inicio` : valor inicial del acumulador (por ejemplo, el "plato" base).
  3. `xs` : lista de insumos o ingredientes que llegarán en secuencia.

  Recursivamente, para cada elemento `x` en `xs`:

  1. Sacar la primera función de *fs*
  2. Aplicarla al acumulador actual ac y al siguiente elemento x de xs.
  3. Guardar el nuevo acumulador en la lista de resultados.
  4. Repetir recursivamente hasta agotar xs.

Ejemplos:

```
;; 1. Suma acumulativa de porciones
>(secuencia-rotacional
(list (lambda (ac x) (+ ac x)))
0
```

```
'(1 2 3 4))
R:'(1 3 6 10)

;; 2. Multiplicacion acumulativa (prefijo de producto)
>(secuencia-rotacional
(list (lambda (ac x) (* ac x)))
1
'(2 3 4))
R:'(2 6 24)

;; 3. Construcción de etiqueta para cada paso
>(secuenciarotacional
(list (lambda (ac x) (string-append ac x "-")))
""
'("pan" "queso" "tomate"))
R:'("pan-" "pan-queso-" "pan-queso-tomate-")
```

P4. Búsqueda del Almacén

Un **robot–buscador** debe internarse en los distintos pasillos de la bodega para recolectar un producto concreto. La bodega se modela como un árbol N-ario: cada lista anidada representa una sala o sub-sala, y los símbolos son nombres de productos. El robot debe registrar la ruta (secuencia de índices) desde la entrada hasta cada aparición del producto buscado.

- **Sinopsis:** `(busqueda-almacen arbol producto)`
- **Característica Funcional:** Listas anidadas, recursión en árboles N-arios, funciones simples, recursión simple.
- **Descripción:**
  1. `arbol`: lista donde cada elemento puede ser un símbolo (producto) o una sublista (subhabitaciones).
  2. `producto`: símbolo que indica el nombre del producto a buscar.
  3. Al recorrer `arbol`, cada vez que se encuentra `producto`, se construye una ruta como lista de índices que muestra por qué sublistas descendió el robot.
  4. Devuelve la lista de todas las rutas encontradas, ordenada de menor a mayor longitud.

Ejemplo:

```
(define bodega
  '(bodega
    (pasillo1 leche pan)
    (pasillo2 (estanteA pan queso) (estanteB manzana pan))
    (pasillo3 fruta)))


;; 1. Buscar ‘‘pan’’
>(busqueda-almacen bodega 'pan)
R:'((1 2)        ; pasillo1 → segundo elemento ‘‘pan’’
  (2 1 1)       ; pasillo2 → estanteA → primer elemento ‘‘pan’’
  (2 2 2))      ; pasillo2 → estanteB → segundo elemento ‘‘pan’’
```

```
;; 2. Buscar ‘‘queso’’
>(busqueda-almacen bodega ’queso)
R:’((2 1 2))      ; pasillo2 → estanteA → segundo elemento ‘‘queso’’

;; 3. Buscar ‘‘fruta’’
>(busqueda-almacen bodega ’fruta)
>’((3 1))         ; pasillo3 → primer elemento ‘‘fruta’’
```

3. Sobre la entrega

- Se deberá entregar un archivo por cada ejercicio con la(s) función(es) solicitadas, con el siguiente formato de nombres:
  - P1.rkt, P2.rkt, P3.rkt, P4.rkt
- Se debe programar siguiendo el paradigma de la programación funcional, no realizar códigos que siguen el paradigma imperativo. Por ejemplo, se prohíbe el uso de for-each. Para implementar las funciones utilice DrRacket. https://racket-lang.org/download/
- Todo código debe contener al principio #**lang scheme**
- Todos los archivos deben ser de la extensión **.rkt**
- Pueden crear funciones que no estén especificadas para utilizar en los problemas planteados, pero solo se revisará que la función pedida funcione y el problema esté resuelto con las características funcionales planteada en el enunciado.
- Cuidado con el orden, comentarios y la indentación de su tarea.
- Las funciones implementadas y que no estén en el enunciado deben ser comentadas de la siguiente forma. SE HARÁN DESCUENTOS POR FUNCIÓN NO COMENTADA
  - Formato para comentar

```
; ; Descripción de la función
; ;
; ; a : Descripción del parámetro a
; ; b : Descripción del parámetro b
```

- Se debe trabajar de forma individual obligatoriamente.
- La entrega debe realizarse en tar.gz y debe llevar el nombre: Tarea4LP_RolAlumno.tar.gz. Ejemplo: Tarea4LP_202500000-0.tar.gz
- El archivo README.txt debe contener nombre y rol del alumno e instrucciones detalladas para la correcta utilización de su programa. De no incluir README se realizará un descuento de 20 puntos.
- La entrega será vía aula y el plazo máximo de entrega es hasta el **Lunes 09 de Junio.**
- Las copias serán evaluadas con nota 0 y se informarán a las respectivas autoridades.
- Solo se responderán consultas sobre la Tarea hasta 48 horas antes de la fecha de entrega.

4. Calificación

4.1. Entrega

Para la calificación de su tarea, debe realizar una entrega con requerimientos mínimos que otorgarán 30 puntos base, luego se entregará puntaje dependiendo de los otros requerimientos que llegue a cumplir.

Entrega Mínima

- Implementar la función 1 (`coincide?`). (10 pts)
- Implementar las dos versiones de recursión para la función 2 (`mezclador-gourmet`). (20 pts — 10 pts por cada variante)

Nota: Las funciones deben devolver respuestas correctas en el formato indicado en sus ejemplos.

Entrega

Luego de cumplir con la Entrega Mínima, puede obtener más puntaje cumpliendo con los siguientes puntos (puede haber puntaje parcial por cada punto):

- Implementación de funciones para la correcta resolución de los casos de prueba (Total 70 pts)
  - Implementar P3 – `secuencia-rotacional`. (30 pts)
  - Implementar P4 – `busqueda-almacen`. (40 pts)

Para todas las funciones, existe puntaje parcial acorde a los casos de prueba que resuelvan.

Descuentos

- Falta de comentarios sobre cada `define` (–5 pts c/u, máx. –20 pts)
- Falta de README (–20 pts)
- Falta de información obligatoria en el README (nombre, rol, instrucciones) (–5 pts c/u)
- Falta de orden o mala indentación en el código (–5 a –20 pts según gravedad)
- Mal nombre en los archivos entregados (–5 pts c/u; –20 pts si es el `.tar.gz`)
- Uso de funciones prohibidas (cada función afectada recibe 0 pts)
- Nombre de función distinta a la especificada (–10 pts c/u)
- Uso de código generado por IA (cada función afectada recibe 0 pts y si se detectan 2 funciones totalmente escritas por IA, se considerará copia y se informará a las respectivas autoridades.)
- Entrega tardía (–10 pts si es dentro de la primera hora; –20 pts por cada día o fracción de retraso)